Aberystwyth University

CS2015-2018 Study Group

CS10720 Printed Material

First Year Module Exam Content

This document is essential material to be printed out and taken into the CS10720 Problems and Solutions open book exam. It's a collaboratively produced document by the 2015 Computer Science study group with 130+ active students. The document summarises the years course content and puts it into an 'exam ready format'.

16th May 2016, Author: wic4. Contributors: aaw13, jam30, pmj5, gjr4, nas16 mol12, mik32,jam86, pit, top10, jos67, haa14, reb28, les35

Contents

1. NUMBERS

1.1 Unsigned & Signed Magnitude.

For this section, you will need a basic understanding of binary conversions. If you still struggle with binary use the binary key found in the Appendix.

Let’s take a number (eg. 3) in binary, which looks like this below:

0011

Three (3) represented in binary.

Signed magnitude tries to solve the problem of representing negative binary numbers. It does this taking the leftmost bit, also known as the *significant bit* (represented in red below):

0 011

and then treating it as a NEGATIVE or POSITIVE sign. This leftmost bit is now called the signed bit.

0 011

'Signed' positive three (3) represented in binary.

When signed, the 0 represents a negative number, meaning that the bits that follows it are all POSITIVE. Were we to represent a negative number we would use a 1 as the *significant bit.* See the example below:

1 011

'Signed' negative three (3) represented in binary.

1.2 One's Complement.

One's complement also uses a *significant bit* at the beginning of a number but its does something slightly different.

It takes all the remaining bits and then "FLIPS" them. This means, where there was a one there is now a zero and where there was a zero there is now a one. See an example below:

1 011 - (negative three [-3] represented in signed binary

1 100 - (negative three [-3] represented in one's compliment

- Negative three (-3) in ones compliment looks like positive four (4) in signed magnitude.
- To convert normal negative binary to one's compliment we sign the first bit and then FLIP ALL THE BITS.

Further examples (One's Compliment)

0 110 (positive six [6] represented in binary)

0 110 (positive six [6] represented in one's compliment)

0 101 (positive five [5] represented in binary)

0 101 (positive five [5] represented in one's compliment)

Please note, as seen above. Positive binary numbers are the same in one's compliment and signed magnitude.

One's Complement Negative numbers

1 101 (negative five [-5] represented in signed magnitude)

1 010 (negative five [-5] represented in one's compliment)

1 010 (negative two [-2] represented in signed magnitude)

1 101 (negative two [-2] represented in one's compliment)

Please note in one compliment we flip all the bits excluding the *significant bit*

1.3 Two's Complement.

Two's complement also uses a *significant bit* at the beginning of a number. If you understand one's compliment [see above] Then two compliment should be easy to understand.

Examples (Two's Compliment)

0 110 (positive six [6] represented in binary signed magnitude)

0 110 (positive six [6] represented in one's compliment)

0 110 (positive six [6] represented in Two's compliment)

0 101 (positive five [5] represented in binary signed magnitude)

0 101 (positive five [5] represented in one's compliment)

0 101 ('Signed by default' positive five [5] represented in Two's compliment)

Note that the Positive representation of binary numbers is the same in signed magnitude, one's compliment as it is in two's compliment.

Two's Complement Negative numbers

To represent negative numbers in two's compliment, we first take the one's compliment number and then add one.

A negative number in two's = the number one's compliment + 1.

1 110 (negative six [-6] represented in binary signed magnitude)

1 001 (negative six [-6] represented in one's compliment)

1 010 (negative six [-6] represented in Two's compliment)

Above, we took the one's compliment value in binary and just added one to give us the two's .

1 101 (negative five [-5] represented in binary signed magnitude)

1 010 (negative five [-5] represented in one's compliment)

1 011 (negative five [-5] represented in Two's compliment)

1.4 Excess Representation (Using a Bias)

Excess Representation is one of the more simple representations when you wrap your head around it. Essentially. In this representation, zero in Binary (0000) is treated as:

0 - *b* *(in simple english, we are saying Zero minus the Bias)*

Where **b** is the bias.

So, if the bias is **12**, then **0000** in Binary is treated as **-12** in Decimal.

To convert a decimal number to Excess Representation, all you have to do is add the bias onto the decimal number, and then convert to binary.

Example, with a bias of 12 and a bit string length of 4:

-12 = 0000 (-12 + 12 = 0)
-11 = 0001 (-11 + 12 = 1)
3 = 1111 (3 + 12 = 15)

As you can see, there is no such thing as a sign bit in Excess Representation, with negative numbers being identified through the use of the bias.

To find the decimal representation of an Excess Representation number, all you have to do is convert the binary string to decimal, and then subtract the Bias.

Here's a few examples, with a bias of 7 and a bit string length of 4:

0000 = (0 - 7 = -7)
0001 = (1 - 7 = -6)
1111 = (15 - 7 = 8)

Again, we see, there is **no such thing as a sign bit** in Excess Representation, this is important to remember, please don't get confused.

So, as said earlier, Excess Representation is pretty easy, as long as you remember to ***ALWAYS*** take the Bias into account.

1.5 Answers to questions around numbers

Q. Is there such a thing as negative zero [-0]

It can be said that 0 == -0. However, the issue with some number representations is that there are multiple ways of printing 0:

- Signed magnitude: 100000 and 000000 both equal zero
- One's complement: 111111 and 000000 both equal zero

These can cause issues and are best avoided, which Two's Complement was intended to avoid by shifting the negative spectrum by one

Q. What are the ranges of numbers we can store in binary.

A. See the table below.

Number of bits	Range (signed) $[2^n - 1]$	Range (unsigned) $\pm[2^{n-1} - 1]$
4	0 through 15	-8 through 7
8	0 through 255	-128 through 127
16	0 through 65,535	-32,768 through 32,767
32	0 through 4.29 billion	-2.15 billion through 2.15 billion

1.6 Binary Hex Chart and Tools

Binary number sequence

1 2 4 8 16 32 64 128 256 512 1024 2048 4096

HEX Position Sequence

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*A*	*B*	*C*	*D*	*E*	*F*
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

HEX Number sequence

16 256 4096

Conversion Chart.

This section comes with an accompanying document. The accompanying document must also be

printed and taken into the exam. It is a key with pre-made of Decimal to Hexadecimal (for Positive numbers) 0 to 255. Binary (Signed magnitude for Negative numbers), One's compliment and Two's complement.

2. IEEE-754 Representation

Quick overview

- Leftmost bit is sign bit, 0 is + 1 is -
- Following 8 digits is Exponent, represents the movement of the floating point.
- Exponent has a bias of -127 so representation must reflect this (an exponent with the binary representation of the value 129 would indicate a movement of 2)
- A positive Exponent shows the point has been moved left, a negative one shows the point has moved right)
- Final 23 digits is the Mantissa
- First digit of mantissa is implied as the mantissa must always be 1.x (in binary)

Format

- +/- exponent implied 1 Mantissa
- 0 0000 0000 (1) 000 0000 0000 0000 0000 0000

Special cases

- 0 has empty Mantissa and Exponent (All 0's). This can be + or -
- Infinity has an exponent of all 1's and a mantissa of all 0's. Can be + or -
- NaN (Not a Number) Exponent of all 1's and a none 0 mantissa (Exact mantissa size is irrelevant as long as it isn't 0)
- Denormalised - Exponent of all 0's, Mantissa none 0
- Denormalised numbers lack the implied leading 1

Decimal to IEEE

- Example for 5.25.
- Translate 5.25 to simple binary floating point = 101.01 (After the decimal point think that each bit is worth half the last, ½, ¼, ⅛ etc. 0.25 is ¼ so after the point we have .01)
- We have 101.01, we need to move the point to just after the first 1, this is 2 movements to the left giving 1.0101.
- In the Mantissa the first bit is implied so remove the 1 from before the point to give 0101.
- The mantissa is 23 bits long so we add the remaining 0's after our value
- We get 010 1000 0000 0000 0000 0000
- The movement of the bit was 2^2, 2 places left. Therefore we have an exponent of +2. Add 127 to this to get 129.
- 129 is 1000 0001. This is an exponent of 2.
- 5.25 is positive so the sign bit is 0
- 5.25 = 0 1000 0001 010 1000 0000 0000 0000 0000
-

IEEE to Decimal

- Essentially the reverse of the above. We start with 1 1000 0001 010 1000 0000 0000 0000 0000
- Remove excess 0's from Mantissa and return the implied 1 gives 1.0101
- Exponent of 1000 0001 is 129. 129 - 127 = 2.
- +2 means we move the point 2 to the right for the original number
- 101.01
- 101.01 is 5.25
- This time the sign bit is 1 so the number is negative
- So Decimal is - 5.25

Remember

- If converting from decimal to IEEE and the decimal is negative ignore the sign when working out the Mantissa the sign bit in the final IEEE output is all that is affected
- In converting a Decimal to IEEE positive Exponent if the floating point is moved to the left when converting to Decimal to floating point negative if it moves right.
- When converting from IEEE back to Decimal remember that a positive Exponent means you move the point right to find the original floating point value and left for a negative.

Q. What is IEEE 754

IEEE 754 is THE standard for Floating-Point Arithmetic within computer programming environments. This was established in 1985 by the IEEE (Institute of Electrical and Electronic Engineers.) This standard helped address many issues which were found within the diverse floating point implementations which the industry used. This standard can be enforced entirely within Software, entirely within the hardware or in any combination of both hardware and software. Most computers used today follow this standard. However this can change depending on the numerical results and exceptions are uniquely determined by the values of the input data, sequence of the operations, and destination formats, which is all under and determined by the user control.

<insert image from noonian soongs document, cannot do it now due to compatibility> Pasi

IEEE 754 Formating

IEEE 754 is formated into The Sign Bit, The Exponent and The Mantissa *(see number representation section below. Pg16 and above)*

Sample Exam Question

The Sign Bit, is either where we represent number being positive or negative with a 0 or a 1. The 0 denotes Positive and 1 denotes a Negative. If these values are flipped it will affect the sign of the Bit.

Answer

The Exponent will cover both Negative and Positive numbers. For this The Exponent requires a Bias, this Bias is added to the “actual” Exponent in order to get or give the “Stored” Exponent. In this instance for IEEE Single-precision floating points, the Bias value will be set or be 127. So within this if an Exponent of zero, would mean 127 is “stored” within the Exponent Field. Yet an example of a “Stored” Exponent - if the stored value is 200 this would indicate an exponent of 200-127 which equals 73. Within Double Precision the Exponent field(area) is different(what a shock) here it is 11 Bits long and will have a bias of 1023. Now onto the final component within IEEE 754’s formating.
- The Mantissa also known as the “significand” This represents the “Precision Bits” of the given number

2. Matrices and Arrays

2.1 Matrix Facts:

- 1 x *n* matrices are called row vectors
- *m* x 1 matrices are called column vectors
- in an *m* x *n* matrix, called *A*, you reference/denote entries in rows (*i*) and columns (*j*) as a_{ij}

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{pmatrix}$$

- *n* x *n* matrix with 1 on diagonal and 0 elsewhere is identity matrix

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Basic Matrix Manipulation:

Addition:

- You can only add together matrices of the same size. You add them by adding each corresponding place in the matrix. Adding Matrix *A* and Matrix *B* would look like:

$$\begin{pmatrix} a_{1,1}+b_{1,1} & a_{1,2}+b_{1,2} & \cdots & a_{1,n}+b_{1,n} \\ a_{2,1}+b_{2,1} & a_{2,2}+b_{2,2} & \cdots & a_{2,n}+b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}+b_{m,1} & a_{m,2}+b_{m,2} & \cdots & a_{m,n}+b_{m,n} \end{pmatrix}$$

Subtraction:

- Identical to addition, but subtracting

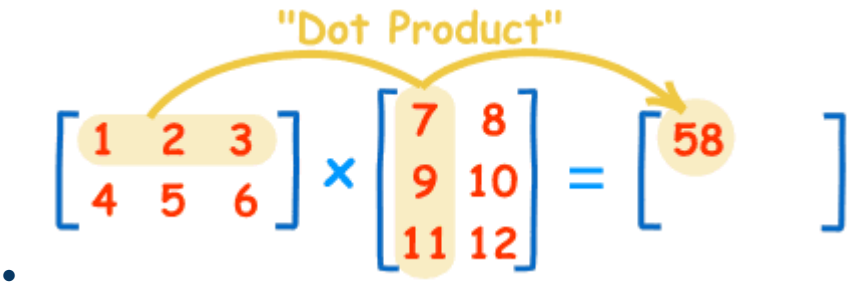
Multiplication:

Scalar Matrix Multiplication (Multiplying a matrix by one number):

- Super easy. Multiply each number with every value in the matrix
-

Multiplying Two Matrices Together:[

- You can multiply two matrices together, regardless of their size. An *m x n* matrix, A, and an *n x o* matrix, B, are multiplied together row by column. Example:

- 

OR

$$\begin{pmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 5 \end{pmatrix} \cdot \begin{pmatrix} 6 & 3 & 1 & 5 \\ 8 & 9 & 7 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 2\cdot 6+4\cdot 8 & 2\cdot 3+4\cdot 9 & 2\cdot 1+4\cdot 7 & 2\cdot 5+4\cdot 2 \\ 1\cdot 6+3\cdot 8 & 1\cdot 3+3\cdot 9 & 1\cdot 1+3\cdot 7 & 1\cdot 5+3\cdot 2 \\ 0\cdot 6+5\cdot 8 & 0\cdot 3+5\cdot 9 & 0\cdot 1+5\cdot 7 & 0\cdot 5+5\cdot 2 \end{pmatrix}$$

$$= \begin{pmatrix} 44 & 42 & 30 & 18 \\ 30 & 30 & 22 & 11 \\ 40 & 45 & 35 & 10 \end{pmatrix}$$

2.1 Uses of Matrices

- Matrices are good for representing objects in space (e.g. an image)
- **Translation** is expressed as addition
- **Scaling** can be expressed as multiplication
- **Rotation** can also be expressed:

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

2.3 Arrays

1 Dimensional Arrays:

- type a [10]
  - Array **a** of type **type** and size **10** occupies a size of **10** consecutive bytes in memory

- **a[8]**
  - gives access to the **9th** item in **a**
- **&a[3]**
  - gives address of 4th item in **a**
- **Type *b = (type *)a;**
  - b gives the starting address of a
- **b+3**
  - gives address of 4th item in a

2.3 Dimensional Arrays:

- Take up a continuous block in memory
- The address of m[i][j] is (m+num)x(i+j)
- Matrices can be represented in an array
- Static arrays are limiting as the length of them needs to be known at compile time
- We can however, use safe upper bound limits, or leave size undetermined until compile time at which we create space when the program is running
  - We can do this using malloc and free

2.4 Malloc - Memory Allocation

Malloc is a lot simpler than you think!
Malloc is used for allocating space that we need freeing up
Free is used for releasing the space

Before using the needed variable, we simply use malloc, in order to allocate space:

```
a = (double *)malloc(sizeof(double)*n); /* allocates space for a */
```

We then use assert to stop the program, if space was not available

```
assert( a!=NULL ); /* stops program if space was not available */
```

Once calculations are complete, we use free to free up space again:

```
free(a); /* frees space used for a again */
```

It is important to remember to use free, as simply deleting pointer variables or ignoring them doesn't free up the memory used.

Quick malloc definitions:

- **assert (condition)**
  - checks if condition is true, if not, then the program is stopped with a useful error message
- **malloc(size)**
  - Returns address of block of size bytes of the memory (or NULL if not available)
  - Needs type cast to appropriate pointer type
- **free (pointer)**
  - Gives back memory indicated by the pointer, to the OS
- **printf (format, argument1, …)**

    - Prints something and includes value of argument in this
- **scanf (format, & argument1,...)**
    - Reads from keyboard
    - Stores value in argument1 (which in this case is the address of argument1)
    - Returns number of inputs
    - Returns negative number if error

2.5 C Code for Matrices

```
#define m 50
#define n 80
#define o 120
double s, a[n][m], b[n][m], c[n][m], d[m][o], e[n][o];
int i, j, k;
/* matrix addition: C = A + B */
for ( i=0; i<n; i++ ) {
        for ( j=0; j<m; j++ ) {
                c[i][j] = a[i][j] + b[i][j];
        }
}

/* scalar multiplication: C = s · A */
for ( i=0; i<n; i++ ) {
        for ( j=0; j<m; j++ ) {
                c[i][j] = s * a[i][j];
        }
}

/* matrix multiplication: E = A · D */
for ( i=0; i<n; i++ ) {
        for ( j=0; j<o; j++ ) {
                e[i][j] = 0; /* initialise sum as 0 */
                for ( k=0; k<m; k++ ) {
                        e[i][j] += a[i][k] * d[k][j];
                }
        }
}
```

2.6 Multi Dimensional Arrays:

When allocating space for these, we have two options

	Pros	Cons
One Malloc (index calcs manually)	● Easy malloc and free ● Very efficient ● Multi-dimen array is just one continuous block of memory anyway	● Index calculations are more effort ● Does not look as natural
Several Mallocs (index calcs automatically	● Looks exactly like static multi-dimen array in use ● Easy to use	● Malloc and free more work ● Less efficient (only a little bit) ● Memory fragmentation possible ● Location in memory unclear

Please note

If you want to find the location of something in a n array that has been initialised by one Malloc allocation and you have defined the array as M [i] [j] you cannot access how you normally would. You need to find the position by
i * <number of 'i's in total> * the number of 'j's you have in total + 'j'

int* m = malloc (size of int * number of 'i's * number of j's)
// You are looking for m [2] [3].
M + 2 * number of i's * number of j's + 3

4. Page Rank

Q. What is Page Rank

Page rank is basically a system that allows web searches to be ordered in a way that is useful to the end user. It covers three main things that users may find important:

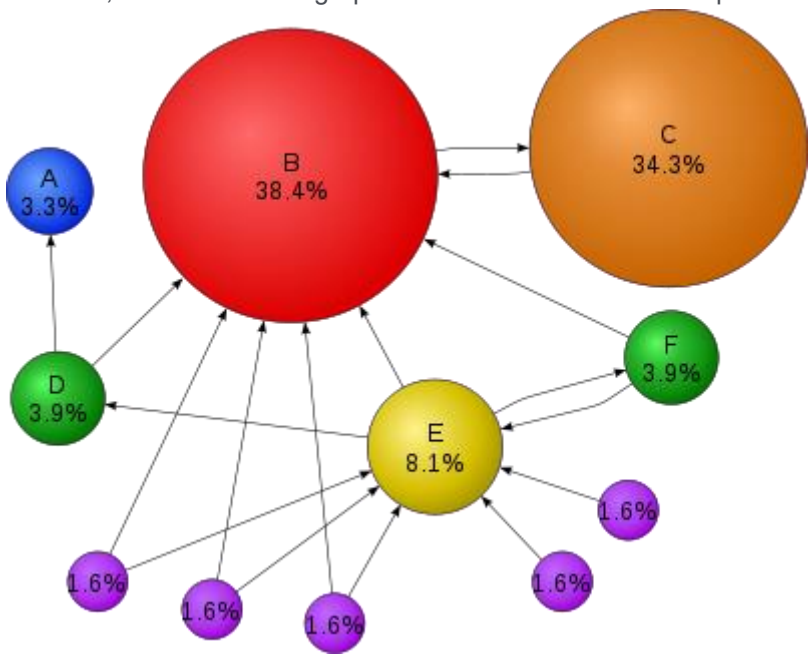
- Importance - what the website means to the end user - is it a blog from Joe who lives in newtown or is it the homepage of youtube?
- Reliability - Is the information on this correct, or is it just random generated text in order to increase hits from search engines
- Authenticity - is the website related to the entity it is about? Google.com is going to be more authentic than google.freewebsitehosting.cc

Q. How does it work

Page rank works by looking at all of the links to a particular website, and determines that if a random person clicks on a random link somewhere, that it would go to that said website. However, the impact

that has depends on the pagerank of the previous website. If the last website has one or two links to it, it is probably unlikely to be Important, Reliable or Authentic.

To understand this easier, then look at this graphic that I didn't steal from wikipedia.



We can see here that site B has a large percentage of the pagerank. This is because it has links to it from lots of different sites. Site C also has a large percentage, even though it is only linked to from one site. This is because site C is linked to from site B, one which has a high page rank. Site D and F have low page ranks, this is because it is only linked to once by a low ranking site. Site A is even lower, because it is only linked to once by site D. The purple sites are the lowers, because they are not linked to from any site.

Q. Is there a really important algorithm that we have to know?

Yes there is

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)},$$

There you go

If that makes no sense to you, I don't really blame you. What it basically means is that the PageRank for page U is calculated by adding up (The pagerank that links to said page, divided by the number of links to the page) for every other page on the internet that there is.

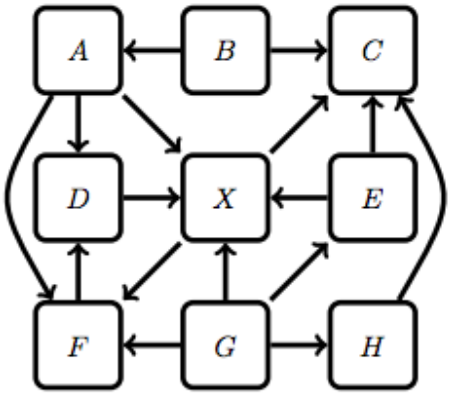
Q. How do you calculate the initial pagerank?

In order to use this algorithm, we need to have a pagerank to start. To address this, we simply count up all of the pages, and have that divided from one.

$1 \div p$ (Where p = the number of pages overall)

Sample Question

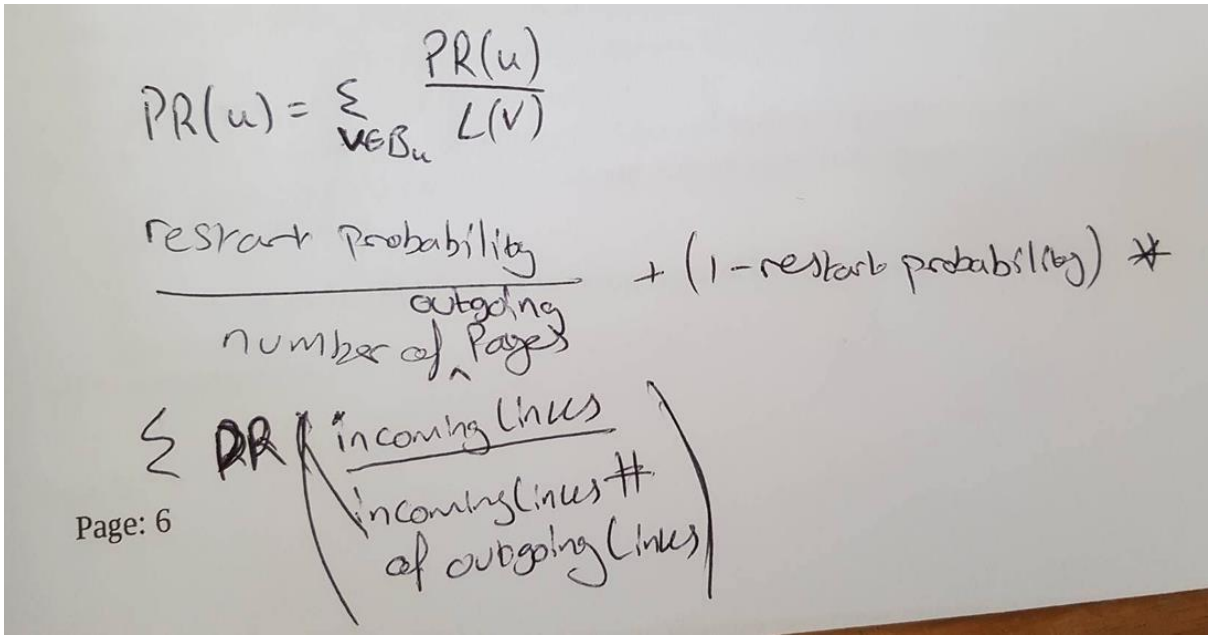
c) For the scenario below (where rectangles represent pages and arrows represent hyperlinks) perform the first step after initialisation to compute the page rank for X. Explain the different components of your calculations. There is no need to actually compute any numbers. It suffices to write down calculations.

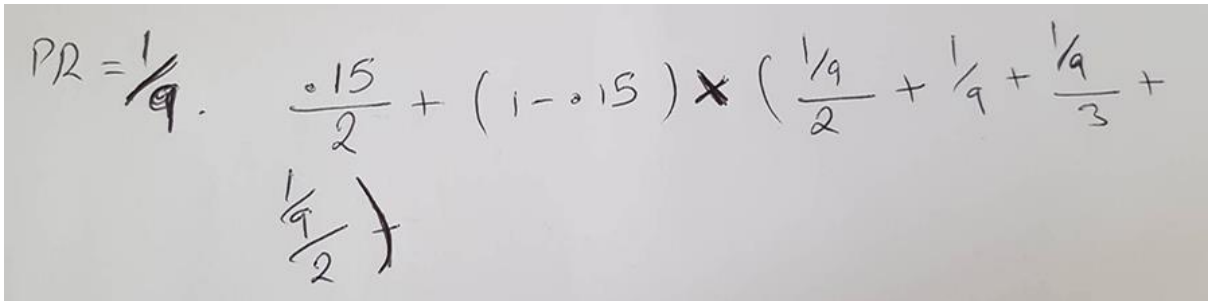


[10 marks]

The Answer to point "C" is

The restart probablilty devided by the total number of pages , all added to (1- restart probability) * the sum of the page rank of each page that links to X / the number of outgoing links from that page\
Restart probability / Total number of pages (Not number of outgoing pages) (Wrong in picture below)





5. Representing Rational Numbers

Q.What is a Floating Point Number

It is any number that has a decimal point in it.

We cannot store the decimal point, We have to represent it with a One or a Zero

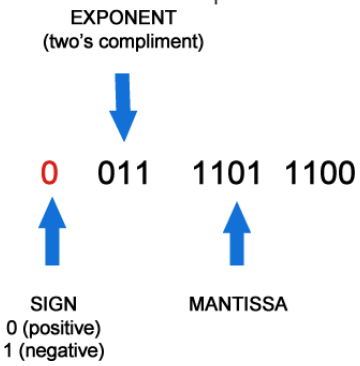
Q. What are types of floating point representation

Binary Floating Point Representation & IEEE 754 Floating Point Representation.

Q.What is the formula for representing floating point numbers.

To solve the problem posed above by the decimal point, we use the formula $M \times B^e$ where M is called the Mantissa, B is the Base ie 2 and e is the Exponent.

Below is an example of how rational numbers represented in binary look.



mantissa will have 23 bits. If the number doesnt have 23 bits, keep adding zeros to the right hand side and make sure you mention that you have leading one dropped off.

6. Computability

DEFINITIONS

A computational problem is defined by a set of finite inputs over a finite input alphabet and for each input a set of correct finite outputs over a finite output alphabet.

A optimisation problem is a computational problem where the output is the value of an optimal solution.

A decision problem is a computational problem where the output is ‘yes’ or ‘no’ (or any similar two answer system - ‘1’ or ‘0’, ‘5v’ or ‘0v’, etc).

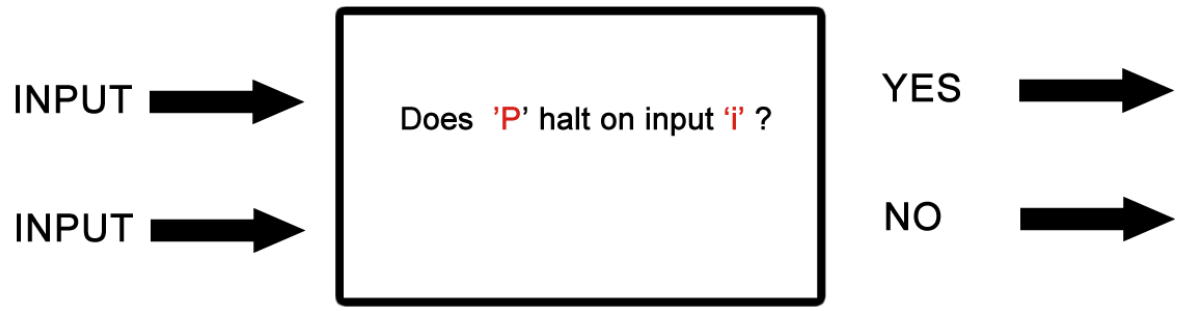
The Problem - In logic we are interested in finding if the premises entail the conclusion?
Premises are the bits you start off with in an argument, they are the bits you know at the beginning or your assumptions and the conclusion is the bit you want to establish.

What we want to know is is there a test or automatic way to find out if they do or don’t. Its also called the decision problem.+

Many Mathematicians worked on trying to find out if first order logic is decidable, this means, can we automatically test whether the premises entail the conclusion

A Program takes some inputs, This could be a question and it give us an answer, yes or no. Example Below.

Lets call the input ’i’ & lets call the program ’P’



if there are any errros on this visual or representation, please Pasi, its creator.

So if we let the computer run indefinitely, would it always be able to give you a yes or no or will it eventually Halt?

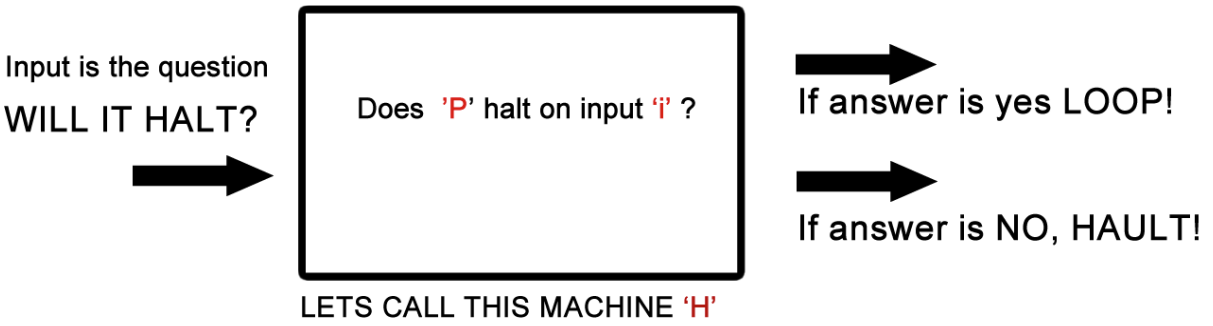
Alan Turing was the first to find out that **first order logic is not decidable.** To prove this he had to show that no program couldn’t decide the answer. This is how he did it…

Let's assume we have a Magical Computer. This magical computer already can solve the Halting problem. Don't worry about how it does this yet but let's assume it does, The next step is it takes the description of a machine and it will give us a simple Yes it Will Halt or NO it won't Halt Answer.

Step 1.

Create if scenarios at end of magic machine 9000 ("H"). If yes Loop, if no hault.

MAGIC MACHINE 9000



if there are any errros on this visual or representation, please Pasi, its creator.

Step 2

Feed H into itself! By this, we mean, simply taking the whole machine and putting it into the input of an identical machine. The new question is "Does H halt on input MAGIC MACHINE 9000 'H'?"

The result will be. If it doesn't halt then it will halt and it it does halt then it won't halt. This is a Paradox! This shows that we started assuming that we could solve the problem, but we ended up with a paradox meaning our assumption was bad

Conclusion: No program can solve the Halting problem

THINGS TO NOTE.

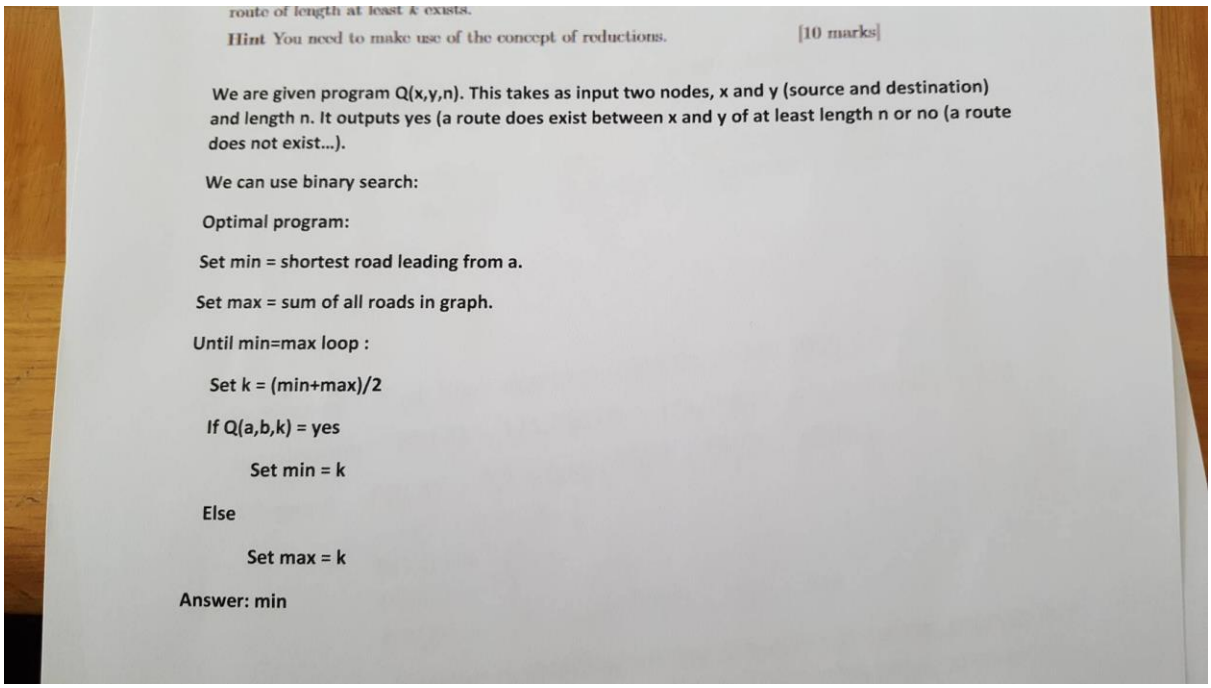
In AI, computers are already in a position where they need to make a decision. We have no idea how or why deep learning works but we just observe that it does.
There are ethical issues when it comes to identifying terrorists when we don't know how the computer comes to the conclusion.

Sample Question Travelling Sales Person / Taxi Driver Problem

c) The traveling salesperson problem (TSP) asks us to find a shortest tour from some home city that visits each of a list of destination cities exactly once at ends at the home city. An instance of this problem is given by a list of cities (including the home city) and for each pair of cities the travel distance between them. We assume that all distances are given by positive integer numbers.

Describe an efficient program (using a clear textual description or pseudo-code) that given a TSP instance computes the optimal length of solution to the TSP and that makes use of a program that for a TSP instance and a number k decides if a tour of length at most k exists. [10 marks]

Answer



route of length at least k exists.

Hint You need to make use of the concept of reductions. [10 marks]

We are given program Q(x,y,n). This takes as input two nodes, x and y (source and destination) and length n. It outputs yes (a route does exist between x and y of at least length n or no (a route does not exist...).

We can use binary search:

Optimal program:

Set min = shortest road leading from a.

Set max = sum of all roads in graph.

Until min=max loop :

Set k = (min+max)/2

If Q(a,b,k) = yes

Set min = k

Else

Set max = k

Answer: min

7. C Code Section.

There is C code for Matrixes in section 2.4

7. Appendix

This section comes with an accompanying document. The accompanying document must also be printed and taken into the exam. It is a key with pre-made of Decimal to Hexadecimal (for Positive numbers) 0 to 255. Binary (Signed magnitude for Negative numbers), One's compliment and Two's complement.

7.2 How to convert between binary, decimal and hex

Please note, you do not need to do conversions, skip to "Gadds Guide" document attached

Binary to Decimal

From right to left, run along the bit pattern. If the bit is a 1, add the related position to the total (remember that each bit is double the value of the bit to the left. For example, consider **100101**).

The first bit is a 1, in the '1' position (as per the binary number sequence above). The total is now 1
The second bit is a 0 - ignore
The third bit is a 1, in the '4' position, so add 4 to the total - now 5
Fourth and fifth bit is 0, so ignore
Finally, the sixth bit is a 1, in the '32' position, so add this to the total - now 37.
100101 in binary is 37 in decimal

Decimal to Binary

From your decimal number, find the next highest number in the binary number sequence above. This will be the starting point. Subtract this number from the number you're translating. This bit will be a 1. Now, go along the binary number sequence. If the next number in the sequence is higher than the subtracted total, the next bit will be 0. If the number is lower, it will be subtractable without making the total negative. As such, subtract this number, and the next bit will be a 1. For example, consider **37**

The next highest number is 32, so this will be our starting point: 1, and 37 - 32 = 5
The next number in the binary sequence is 16. As this is too high, the next bit is 0
8 is also too high, so the third bit is also 0 - so far, we have 100
The next number is 4. As 4 is less than 5, the next bit will be 1, and subtract this from your total (5 - 4 = 1)
2 is too high, so the next bit is 0
Finally, 1 = 1, so the next bit is a 1, giving us the completed bit sequence as **100101**

Don't forget to add leading one's to match what the question is asking for.
If the question asks for an 8 bit representation (as is common being a byte) the bits would be **00100101**

Decimal to Hexadecimal

From your decimal number, find the next highest number in the hexadecimal number series. Divide your starting number

Hexadecimal to Decimal

Binary to Hexadecimal & Hexadecimal to Binary

The simplest way is to translate into decimal first, and then into hex, as per the instructions above. As such, I'm not going to write an explanation. Not sorry.

END

Author

Pasi Chi - Document, Index, Numbers Section, Rational Numbers, Computability Section

The information provided on this and other pages by me, Pasi Chidziva wic4@aber.ac.uk and the contributors listed at the end of this document, is under my own personal responsibility and not that of the Aberystwyth University. Similarly, any opinions expressed are my own and are in no way to be taken as those of Aberystwyth University.

Contributors & Co Authors

Aaron Walker- Unsigned and signed magnitude, ones and twos compliment, Page Rank, Gadds Guide.
Harry Adams - Converting between binary, decimal, and hexadecimal
Jack Morgan -
Pip Turner - Matrices and Arrays/Malloc
Morgan Lang - Brief information on IEEE 754
Lauren Brender - Vague attempts at proofreading
Tommy Pugh - Good man.
Reece Browne - Streamlining IEEE
Lewis Sweeney - Excess Representation, Bits of proof reading